telecoms
our solutions. your success

SS-68BB
MINI BUFFER

User Manual

Revision History

Revision 01	Original document	2004
Revision 02	Cosmetic Update	13 March 2008
Revision 03	Add Appendix A	10 December 2008

CONTENTS

1. INTRODUCTION 3

2. FEATURES 3

3. CONNECTIONS 3
3.1 PC Connection..... 3
3.2 PABX Connection 3

4. PROGRAMMING INSTRUCTIONS 4

5. TECHNICAL SPECIFICATIONS 5

6. CONTACT DETAILS..... 6

APPENDIX A: USING SS-68BB MINI BUFFER AS A SECURITY DONGLE..... 7

1. INTRODUCTION

The **SS-68bb Mini Buffer** is a call logging buffer that is supplied fitted standard with 128 Kbytes of RAM.

Battery backup will allow logging in the event of power failure.

2. FEATURES

Line powered.

PABX connection OPTO-COUPLED

Battery backed operation.

PABX power used when PC is turned off

Full functionality during battery operation.

Beeper warning when buffer full.

Compact.

Easy installation.

LED buffer status indications.

3. CONNECTIONS

3.1 PC Connection

PC Connection is by means of a female 9-way 'D' type connector. The cable to connect the **Mini Buffer** to a PC is a simple '1 to 1' cable, with only pins 2, 3, 4, 5 and 7 being used.

PIN #	PIN DESCRIPTION
2	Transmit to PC
3	Receive from PC
4	Handshake from PC
5	Ground
7	Power (to pin 7 on PC)

3.2 PABX Connection

PABX Connection is by means of a male 9-way 'D' type connector

PIN #	**PIN DESCRIPTION**
2	Data receive from PABX
4 & 7	Handshake to PABX
5	Common 'Ground' signal.
6 & 8	Handshake from PABX

4. PROGRAMMING INSTRUCTIONS

Instruction	Description
S<enter>	Requests a record in ASCII
K<enter>	Requests a compressed record
N<enter>	Advance record pointer. (Compressed mode only)
DMP<enter>	Dumps all stored data
SC nn<enter>	Sets Minimum Record Length. (Default = 0)
S7 n<enter>	7 Bit operation if n = 1. Note: use for 7 bit only operation, do not use for 7 bit with parity
SB nn<enter>	Sets PABX baud rate, where nn represents the first two digits of the baud rate. Max 9600bps
SP nn<enter>	Sets PC baud rate, where nn represents the first two digits of the baud rate. Max 28800bps. **Note:** The new settings will be operational after the buffer is reset.
SD n<enter>	n = 1 enable Date/Time insertion, n = 0 disable Date/Time insertion.
YY n<enter>	Store Year if n = 1
YC n<enter>	Store Century if n = 1 and n = 1 for YY
T YYMMDDhhmm<enter>	Sets the time
SA n<enter>	n = 1 Enable AutoDump; n = 0 Disable AutoDump
SAT nn<enter>	Sets AutoDump timer to nn*100 milliseconds
CLR<enter>	Clears the memory and reset
SYS<enter>	Displays system information
HI<enter>	Output Product Info, Maximum Memory Capacity, Used number of lines used
DR n<enter>	Text on reset: n = 0 Enable, n = 1 Disable
SQ n<enter>	Enables sequence number storing if n = 1
SH nn<enter>	nn = 00 Keep Handshaking on until the buffer is full. (Rev 1.10 and above) nn = 01 to 63 Normal operation were the cycle time is 1 to 63 seconds with a 25% duty factor
SAO nn<enter>	nn = AutoDump Auto Off Time in seconds When set this will stop response to SA n commands and it will cause the AutoDump mode to be cleared when the handshaking has been off for more than the set time. AutoDump then has to be re-enabled using SA 1
V<enter>	Displays Software Version number
TM<enter>	Does a memory test

5. TECHNICAL SPECIFICATIONS

Dimensions	
Housing	Black plastic 130 x 68 x 29 mm
LED indicators	PC Tx & Rx, 80% & 50% buffer full, PABX DATA, Heart beat.
Connectors	Connection to PC: 9 way D-type female Connection to PABX: 9 way D-type male
Storage medium	Battery backed RAM buffering
Storage capacity	128K - 4000 call records
Setup storage	Setup data is stored in non-volatile EEPROM
Compatibility	Compatible with most PABX's units
Required voltage	5V to 12V DC from PC handshaking lines
Current consumption	5 mA to 10mA
Battery backup	3.6V 60mA Ni-Cad. Powers unit up to 2 days – Indefinite if PABX supplies power
PABX baud rate	Software settable (Bd) 300, 600, 1200, 2400, 4800, 9600
PC baud rate	Software settable (Bd) 300, 600, 1200, 2400, 4800, 9600,19200
Data storage	Time stamping of call records and events Data compression, typical ratio 2.5:1 Warning beeper at 95% full DTR enabled dumping of records ASCII handshake protocol Fully error corrected proprietary protocol

6. CONTACT DETAILS

Office:

23 Botha Avenue
Lyttelton Manor
Pretoria, Gauteng
South Africa
Tel: +27 12 664 4644
Fax: +27 86 614 5625 E-mail: info@sstelecoms.com

Postal address:

Postnet Suite 48
Private Bag x 1015
Lyttelton, 0140
Pretoria, Gauteng
South Africa

Sales Support:

South Africa E-mail: sales@sstelecoms.com

United Kingdom E-mail: uksales@sstelecoms.com

Technical Support: E-mail: support@sstelecoms.com

APPENDIX A: USING SS-68BB MINI BUFFER AS A SECURITY DONGLE

INTRODUCTION

With the proliferation of software solutions it is becoming commonplace for software to be pirated or used without authorisation.
A fairly standard solution is to provide a 'dongle' (hardware device) that attaches to the PC and the software will not run if the device is not present. This obvious lock is often a source of irritation to the end user.
A more elegant solution is to provide a 'useful' device that acts as the security dongle. The data buffer is an obvious candidate for the task where a Telephone Management System is the application.
SS Telecoms have developed a simple protocol that will allow software to verify that it is attached to a matching buffer.

DISCUSSION ON SECURITY ISSUES

There is a simple level of security provided by having a data buffer in the first place.
Pitfalls can be identified such as
Another hardware manufacturer can develop a buffer with a similar protocol. This means that it will be possible to hijack the software package by using these alternate buffers
Another software vendor can adapt his software to work with the buffers and thus take over the installed base of buffers allowing this vendor a much cheaper entry to this customer than the original vendor
Solution to issues
The basic solution to the problems outlined above has already needed addressing in the GSM arena. Authentication of users is vital to ensure that billing is accurate, for example. How do they do this? It is done using a system known as 'Challenge / Response' authentication. This will be outlined briefly below.

Challenge Response Description
The PC software and the Buffer have both loaded with a secret key. This key can be up to 16 characters
The PC sends the buffer a challenge consisting of a short randomly generated message of up to 10 characters.
The Buffer performs a 'hashing' function on this message using the secret key as part of the process. It is not possible using a small number of messages to determine what the key is.
The buffer then sends the result of the 'hashing' function to the PC.
The PC then compares this to the result of internally generating the 'hash' result
If the results match, then the buffer is authenticated and the software will run.

IMPLEMENTATION

Setting the key
Use the command '~ nnkk<enter>' where '~' is the tilde character, followed by a space character and then the 2 digit hexadecimal offset ('nn') for the key character which is also sent hexadecimal ('kk') and <enter> is the Carriage Return character 0x0D. The sequence to set the secret code to 'TEST' is shown below:
~ 0054
~ 0145
~ 0253
~ 0354

The process has been automated and the code can be set and tested using the 'Buffer Lock' tool.

Challenging the buffer

The command to challenge the buffer is: 'CODErrr..rrr<enter>' where 'CODE' is the command, 'rrr..rrr' is the random challenge string up to 10 characters in length and <enter> is the Carriage Return character 0x0D.

Buffer response

The buffer will respond with the following:
CODE: hhhh<enter>
Where 'CODE: ' is the response text and 'hhhh' is a 16 bit CRC for the submitted challenge and the secret key.

Response validation

In the code below the variable CRC is a 16 bit unsigned integer and char is 8 bit. CmdBuf is the input buffer where the string starting with 'CODE: ' is stored and CmdPtr indexes the 1st 'h'.

```
void CalcCrc(unsigned char ser_data)
{
crc  = (unsigned char)(crc >> 8) | (crc << 8);
crc ^= ser_data;
crc ^= (unsigned char)(crc & 0xff) >> 4;
crc ^= (crc << 8) << 4;
crc ^= ((crc & 0xff) << 4) << 1;
crc &= 0xFFFF;
}

void mfCode(void)
{
char * p = &CmdBuf[CmdPtr];
int retCrc = HexToInt(p,4); //convert the hex data to int
crc = 0xFFFF;               //initialise the crc
//======== do the calculation on the challenge ==============
for (unsigned int i = 0; i < strlen(test); i++)
{
     CalcCrc(test[i]);
}
//======== do the calculation on the UserKey   ==============
for (int i = 0; i < strlen(SecretKey); i++)
{
     CalcCrc(SecretKry[i]);
}
//==== crc should be equal to retCrc if all is well ==========
}
```